第4-5章 串、数组和广义表

杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

第2章 线性表

第3章 栈和队列

线性结构

可表示为： $(a_1, a_2, \ldots\ldots, a_n)$

第4-5章 串、数组和广义表

操作受限的线性表

内容受限的线性表

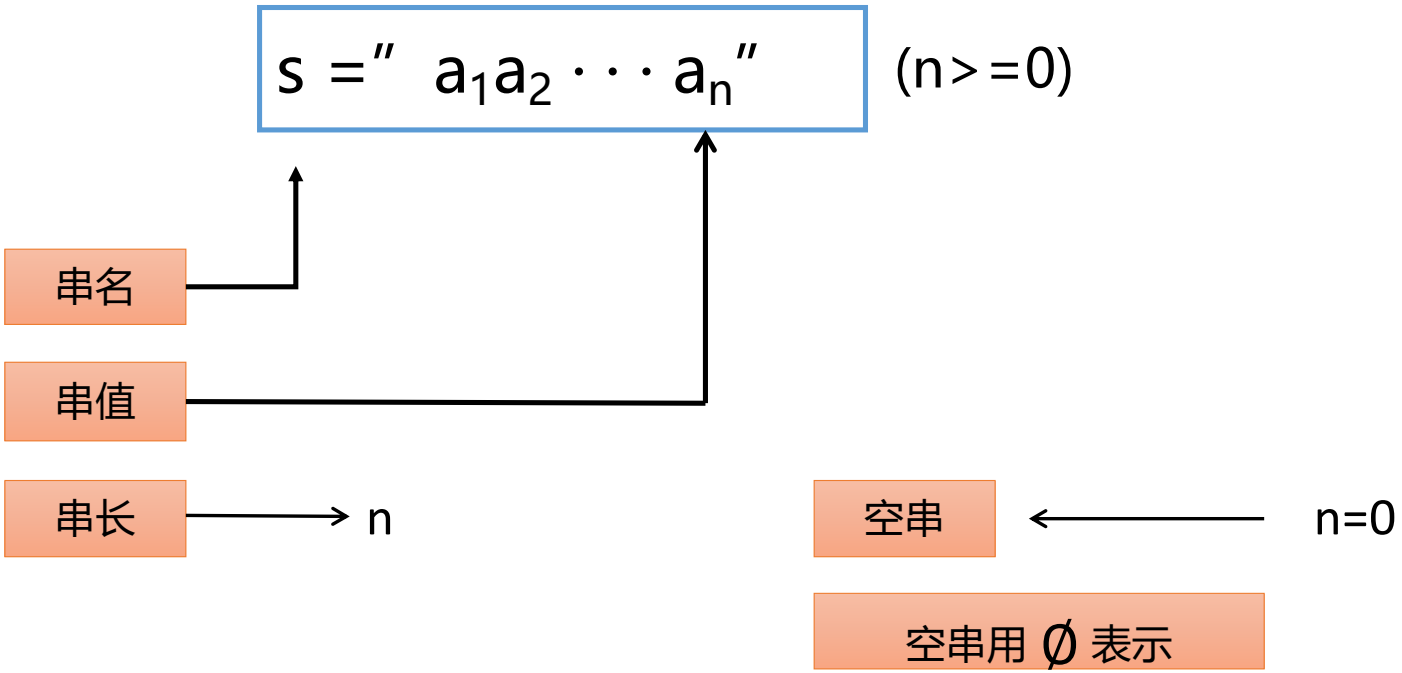
线性结构的推广

第4-5章 串、数组和广义表

杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

4.1 串的定义

串(String)----零个或多个任意字符组成的有限序列



4.1 串的定义——几个术语

杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

- 子串：串中任意个连续字符组成的子序列称为该串的子串

4.1 串的定义

子串：一个串中任意个连续字符组成的子序列（含空串）称为该串的子串。

例如，“abcde”的子串有：

“”、“a”、”ab”、”abc”、”abcd”和”abcde”等

真子串是指不包含自身的所有子串。

4.1 串的定义——几个术语

- 子串：串中任意个连续字符组成的子序列称为该串的子串
- 主串：包含子串的串相应地称为主串
- 字符位置：字符在序列中的序号为该字符在串中的位置
- 子串位置：子串中第一个字符在主串中的位置
- 空格串：由一个或多个空格组成的串，与空串不同

例：字符串a、b、c、d
a= "BEI"
b= "JING"
c= "BEIJING"
d= "BEI JING"

它们的长度是：3 4 7 8
c的子串是：a b
d的子串是：a b
a在c中的位置是：1　a在d中的位置是：1
b在c中的位置是：4　b在d中的位置是：5

4.1 串的定义

杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

串相等：当且仅当两个串的长度相等并且各个对应位置上的字符都相同时，这两个串才是相等的。

例如：

“abcd” ≠ “abc”

“abcd” ≠ “bcde”

- 所有空串是相等的。

第4章 串、数组和广义表

杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

4.1 串的定义

4.2 案例引入

4.3 串的类型定义、存储结构及其运算

5.1 数组

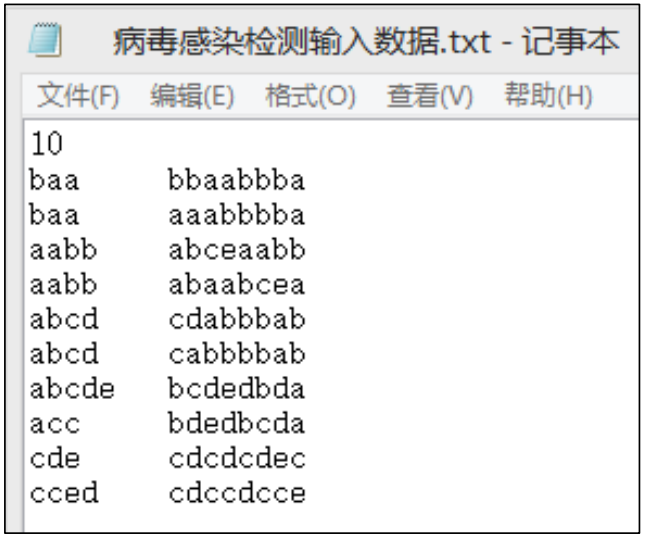
5.2 广义表

4.2 案例引入

串的应用非常广泛，计算机上的非数值处理的对象大部分是字符串数据，例如：文字编辑、符号处理、各种信息处理系统等等。

案例4.1：病毒感染检测

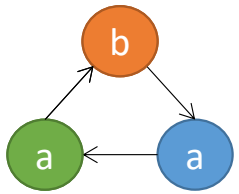
- 研究者将人的DNA和病毒DNA都表示成由一些字母组成的字符串序列



4.2 案例引入

- 然后检测某种病毒DNA序列是否在患者的DNA序列中出现过，如果出现过，则此人感染了该病毒，否则没有感染。
  - 例如：假设病毒的DNA序列为baa，患者1的DNA序列为aaabbba，则感染，患者2的DNA序列为babbba，则未感染。

(注意: 人的DNA序列是线性的，而病毒的DNA序列是环状的)



baa
aab
aba

病毒感染检测输出结果.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

baa	bbaabbba	YES
baa	aaabbbba	YES
aabb	abceaabb	YES
aabb	abaabcea	YES
abcd	cdabbbab	YES
abcd	cabbbbab	NO
abcde	bcdedbda	NO
acc	bdedbcda	NO
cde	cdcdcdec	YES
cced	cdccdcce	YES

第4章 串、数组和广义表

杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

4.3.1 串的抽象类型定义

ADT String {

数据对象：$D=\{a_i \mid a_i \in CharacterSet, i=1,2,\cdots,n, n\geqslant 0\}$

数据关系：$R1=\{ <a_{i-1}, a_i> \mid a_{i-1}, a_i \in D, i=2,\cdots,n\}$

基本操作：

(1) StrAssign(&T,chars) //串赋值

(2) StrCompare(S,T) //串比较

(3) StrLength(S) //求串长

(4) Concat(&T,S1,S2) //串连结

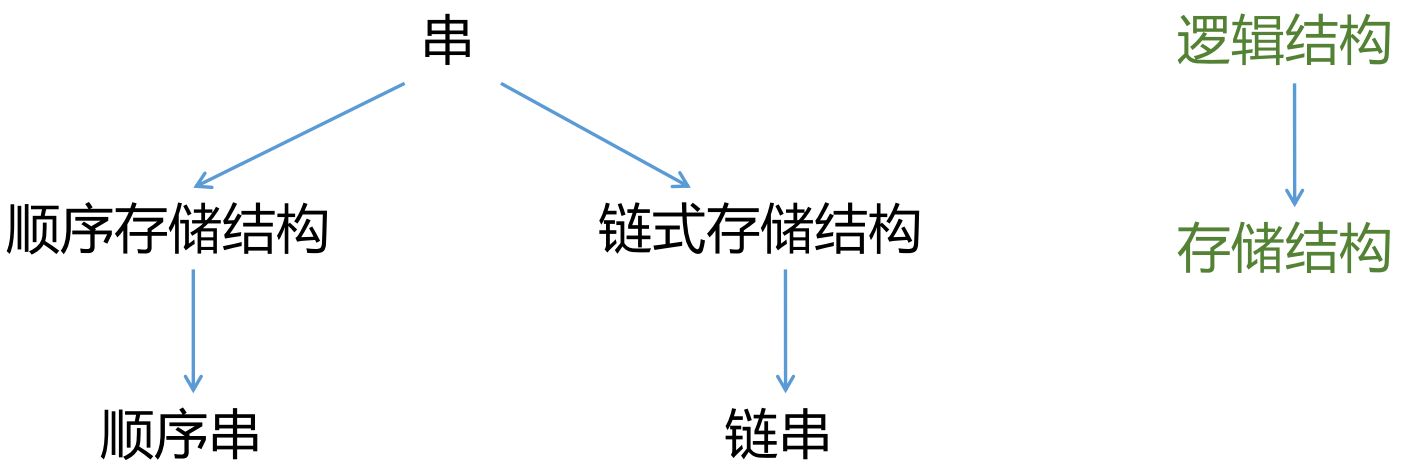
4.3.1 串的抽象类型定义

杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

```
(5) SubString(&Sub,S,pos,len)      //求子串
(6) StrCopy(&T,S)                  //串拷贝
(7) StrEmpty(S)                    //串判空
(8) ClearString(&S)                //清空串
(9) Index(S,T,pos)                 //子串的位置
(10) Replace(&S,T,V)               //串替换
(11) StrInsert(&S,pos,T)           //子串插入
(12) StrDelete(&S,pos,len)         //子串删除
(13) DestroyString(&S)             //串销毁
}ADT String
```

4.3.2 串的存储结构

串中元素逻辑关系与线性表的相同，串可以采用与线性表相同的存储结构。



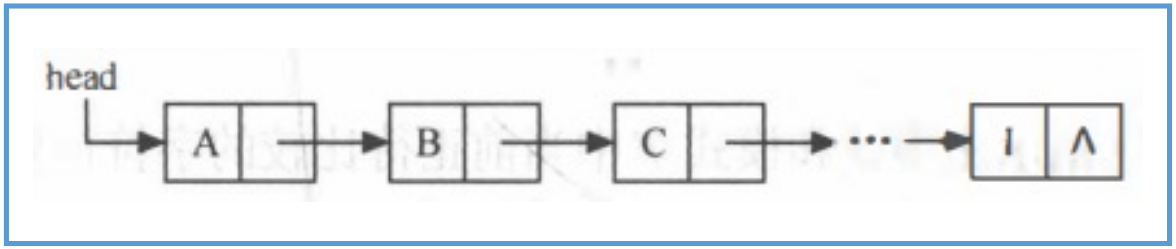
4.3.2 串的存储结构

- 串的顺序存储结构

```
#define MAXLEN 255
typedef struct{
        char ch[MAXLEN+1];   //存储串的一维数组
        int length;          //串的当前长度长度
}SString;
```

4.3.2 串的存储结构

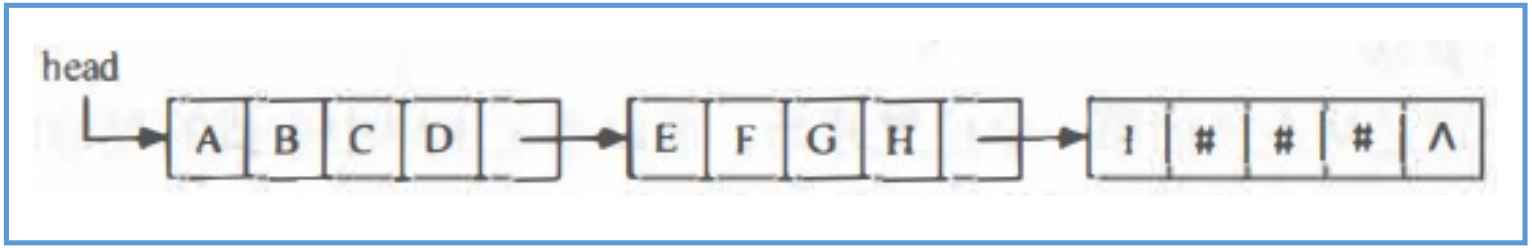
- 串的链式存储结构



优点：操作方便　　缺点：存储密度较低

$$\text{存储密度} = \frac{\text{串值所占的存储}}{\text{实际分配的存储}}$$

可将多个字符存放在一个结点中，以克服其缺点



4.3.2 串的存储结构

- 串的链式存储结构----块链结构

```
#define CHUNKSIZE 80          //块的大小可由用户定义
typedef struct Chunk{
    char ch[CHUNKSIZE];
    struct Chunk *next;
}Chunk;

typedef struct{
    Chunk *head,*tail;        //串的头指针和尾指针
    int curlen;               //串的当前长度
}LString;                     //字符串的块链结构
```

4.3.3 串的模式匹配算法

- 算法目的：

  确定主串中所含子串（模式串）第一次出现的位置（定位）

- 算法应用：

  - 搜索引擎、拼写检查、语言翻译、数据压缩

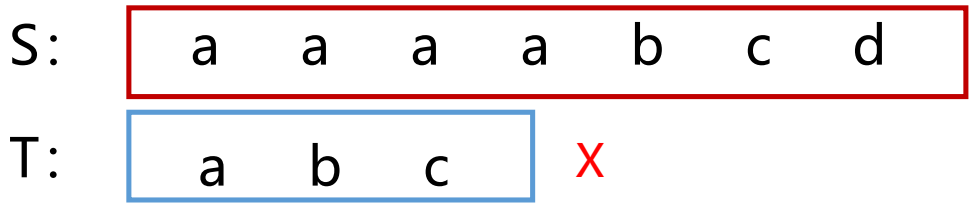
- 算法种类：

  - BF算法（Brute-Force，又称古典的、经典的、朴素的、穷举的）
  - KMP算法（特点：速度快）

4.3.3 串的模式匹配算法----BF算法

杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

Brute-Force简称为BF算法，亦称简单匹配算法。采用穷举法的思路。



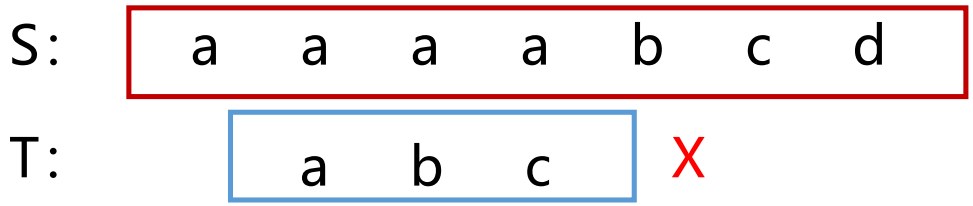
主串：正文串

子串：模式串

4.3.3 串的模式匹配算法----BF算法

杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

Brute-Force简称为BF算法，亦称简单匹配算法。采用穷举法的思路。



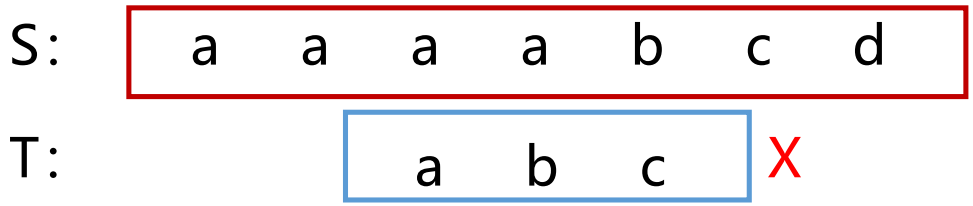
主串：正文串

子串：模式串

4.3.3 串的模式匹配算法----BF算法

杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

Brute-Force简称为BF算法，亦称简单匹配算法。采用穷举法的思路。



主串：正文串

子串：模式串

4.3.3 串的模式匹配算法----BF算法

杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

Brute-Force简称为BF算法，亦称简单匹配算法。采用穷举法的思路。

S: a a a a b c d　　主串：正文串

T: a b c　√ 匹配成功　子串：模式串

算法的思路是从S的每一个字符开始依次与T的字符进行匹配

4.3.3 串的模式匹配算法----BF算法

杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

例如，设目标串S=“aaaaab”，模式串T=“aaab”。

S的长度为n(n=6)，T的长度为m(m=4)。

BF算法的匹配过程如下：

i=1, j=1

S: a a a a a b

T: a a a b

4.3.3 串的模式匹配算法----BF算法

杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

例如，设目标串S=“aaaaab”，模式串T=“aaab”。

S的长度为n(n=6)，T的长度为m(m=4)。

BF算法的匹配过程如下：

i=1, j=1

S: a a a a a b

T: a a a b

4.3.3 串的模式匹配算法----BF算法

杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

例如，设目标串S=“aaaaab”，模式串T=“aaab”。

S的长度为n(n=6)，T的长度为m(m=4)。

BF算法的匹配过程如下：

i=1, j=1

S: a a a a a b

T: a a a b

4.3.3 串的模式匹配算法----BF算法

杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

例如，设目标串S=“aaaaab”，模式串T=“aaab”。

S的长度为n(n=6)，T的长度为m(m=4)。

BF算法的匹配过程如下：

i=1, j=1

S: a a a a a b

≠

T: a a a b

i

j

匹配失败：

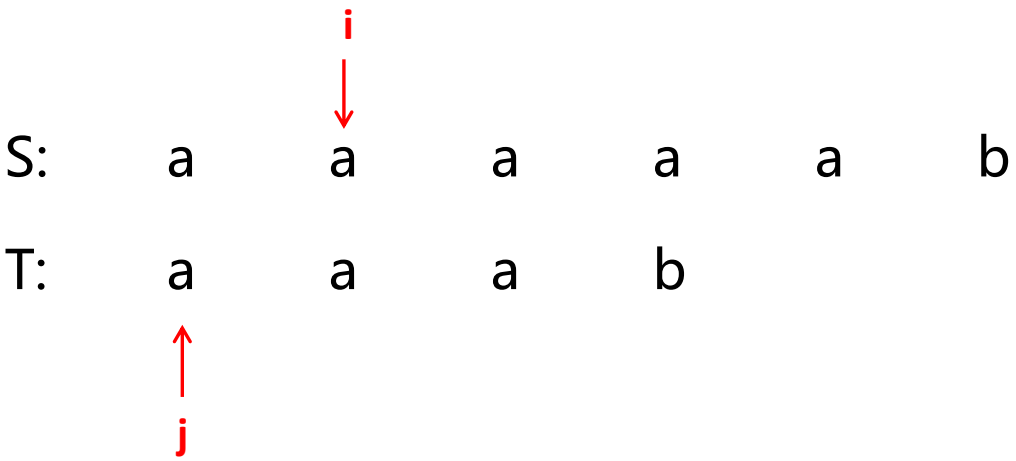
i=i-j+2=2 (回溯)

j=1 (从头开始)

4.3.3 串的模式匹配算法----BF算法

杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

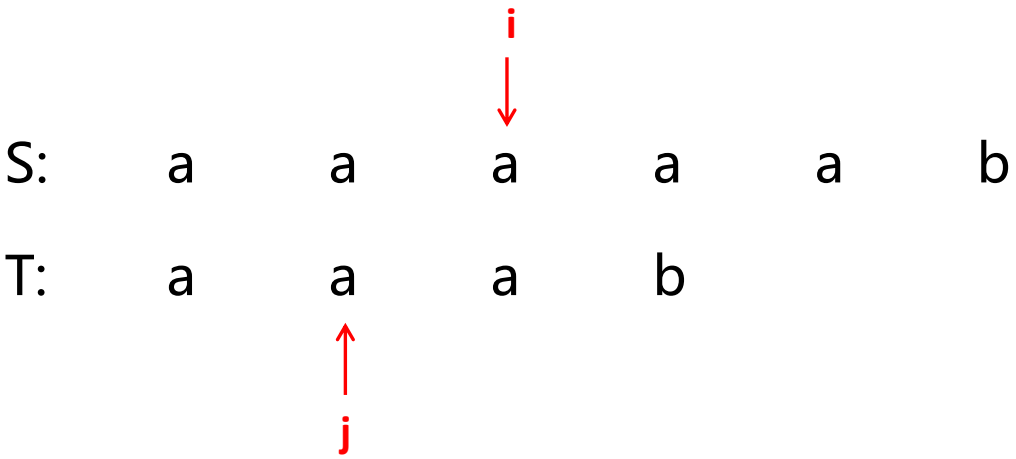
i=2, j=1



4.3.3 串的模式匹配算法----BF算法

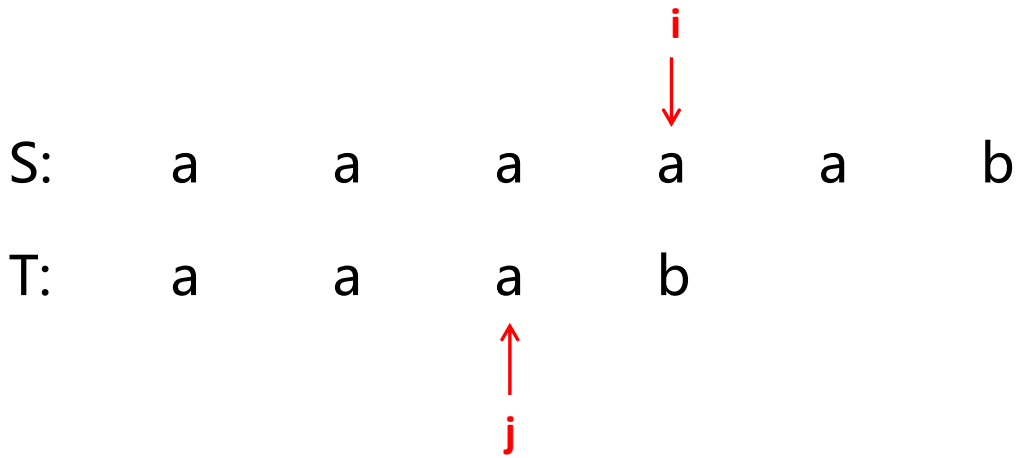
杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

i=2, j=1



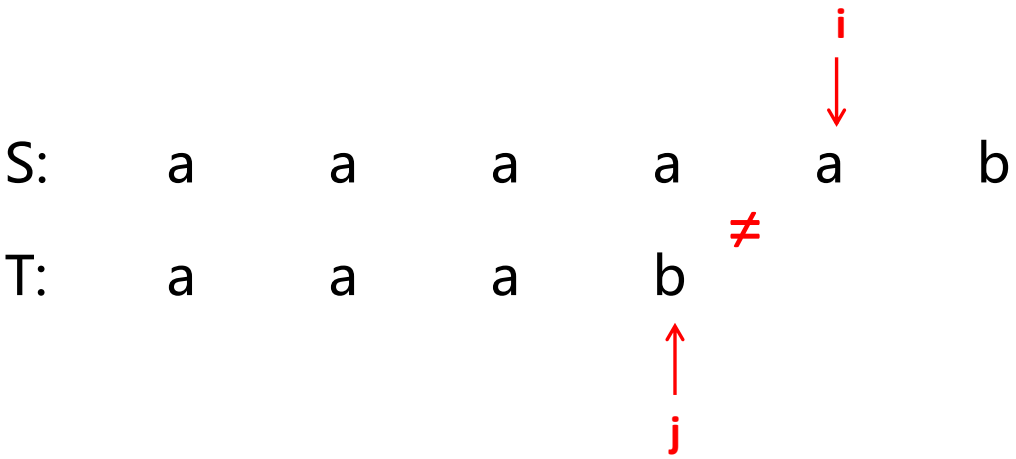
4.3.3 串的模式匹配算法----BF算法

i=2, j=1



4.3.3 串的模式匹配算法----BF算法

i=2, j=1



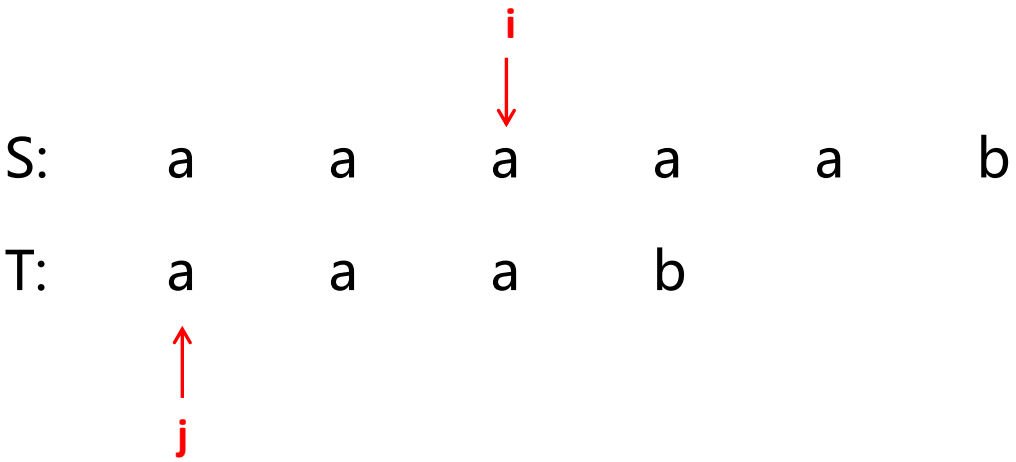
匹配失败：

i=i-j+2=3 (回溯)

j=1 (从头开始)

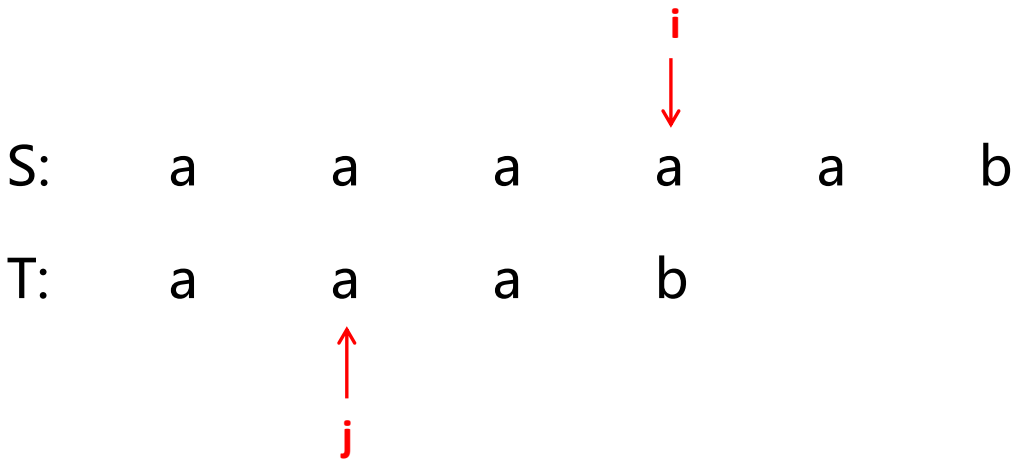
4.3.3 串的模式匹配算法----BF算法

i=3, j=1



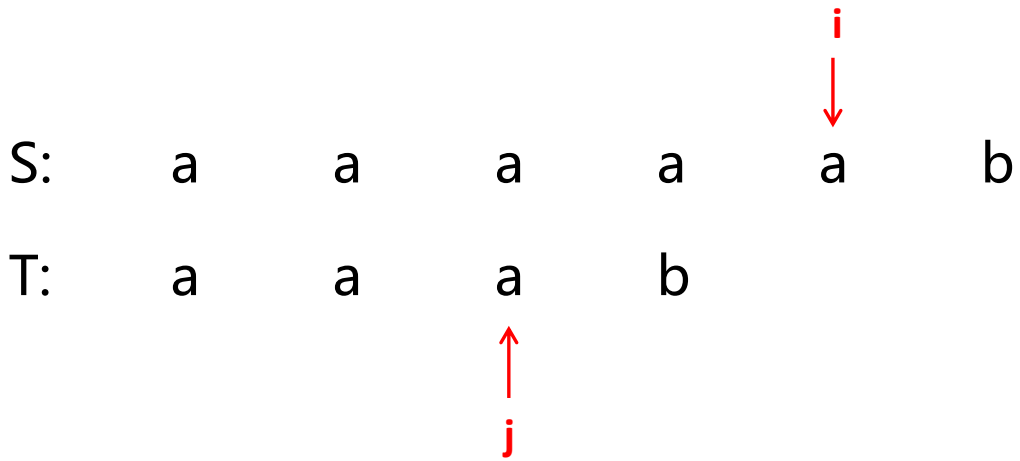
4.3.3 串的模式匹配算法----BF算法

i=3, j=1



4.3.3 串的模式匹配算法----BF算法

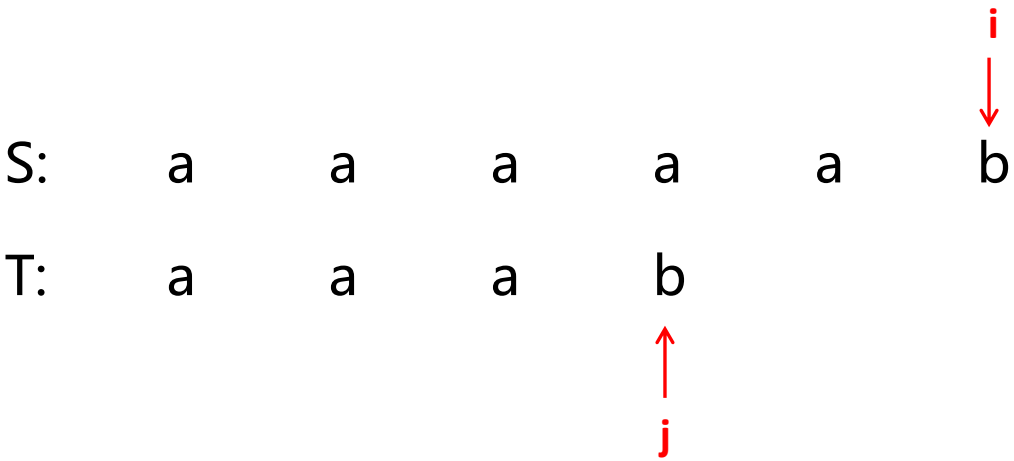
i=3, j=1



4.3.3 串的模式匹配算法----BF算法

杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

i=3, j=1



4.3.3 串的模式匹配算法----BF算法

杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

i=3, j=1

S: a a a a a b

T: a a a b

匹配成功：

i=7， j=5　　返回 i - T.length = 3

BF算法设计思想

Index(S,T,pos)

- 将主串的第pos个字符和模式串的第一个字符比较
  - 若相等，继续逐个比较后续字符;
  - 若不等，从主串的下一字符起，重新与模式串的第一个字符比较。
- 直到主串的一个连续子串字符序列与模式串相等。返回值为S中与T匹配的子序列第一个字符的序号，即匹配成功。
- 否则，匹配失败，返回值 0

BF算法描述

杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

```
int Index_BF(SString S, SString T ){
    int i=1, j=1;
    while (i<=S.length && j<=T.length) {
        if (s.ch[i]==t.ch[j]){++i;++j; }   //主串和子串依次匹配下一个字符
        else {i=i-j+2; j=1;}               //主串、子串指针回溯重新开始下一次匹配
    }
    if (j>T.length) return i-T.length ;    //返回匹配的第一个字符的下标
    else return 0;                         //模式匹配不成功
}
```

BF算法描述

杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

```
int Index_BF(SString S, SString T, int pos ){
    int i=pos, j=1;
    while (i<=S.length && j<=T.length) {
        if (s.ch[i]==t.ch[j]){++i;++j; }   //主串和子串依次匹配下一个字符
        else {i=i-j+2; j=1;}               //主串、子串指针回溯重新开始下一次匹配
    }
    if (j>T.length) return i-T.length ;    //返回匹配的第一个字符的下标
    else return 0;                         //模式匹配不成功
}
```

BF算法时间复杂度

例： S=“0000000001”，T=“0001”，pos=1

若n为主串长度，m为子串长度，最坏情况是

√主串前面n-m个位置都部分匹配到子串的最后一位，即这n-m位各比较了m次

√最后m位也各比较了1次

总次数为: (n-m)*m+m =(n-m+1)*m

若m<<n，则算法复杂度O(n*m)

KMP (Knuth Morris Pratt)算法

杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

KMP算法是D.E.Knuth、J.H.Morris和V.R.Pratt共同提出的，简称KMP算法。

该算法较BF算法有较大改进，从而使算法效率有了某种程度的提高。

KMP算法设计思想

杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

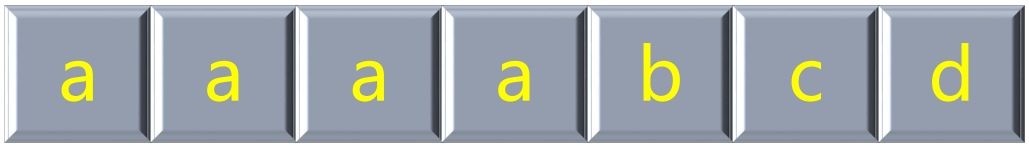
利用已经部分匹配的结果而加快模式串的滑动速度

且主串S的指针i不必回溯! 可提速到O(n+m)!

串的模式匹配——KMP算法

- BF算法思路：穷举法

主串：
人的基因



子串：
病毒基因

x x x √ 匹配成功

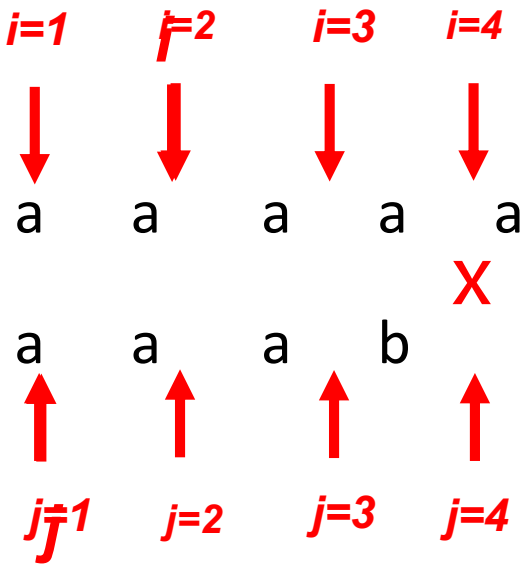
- KMP算法在BF算法的基础上进行加速

串的模式匹配——KMP算法

给定人的基因=“aaaaab”，病毒的基因=“aaab”

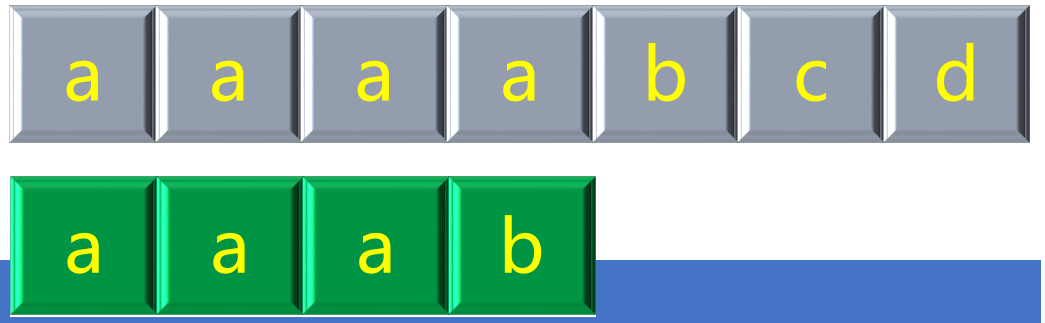
BF算法的匹配过程如下：

第一轮：



$i = 1$

$j = 1$



串的模式匹配——KMP算法

杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

第二轮：

i=2 i=3 i=4 i=5

人的基因：a a a a a b c d

X

病毒基因：a a a b

j=1 j=2 j=3 j=4

指针可以不回溯么？

$i = 5 \quad j = 4$

$i = i - (j - 1) = i - j + 1 \qquad +1 = i - j + 2$

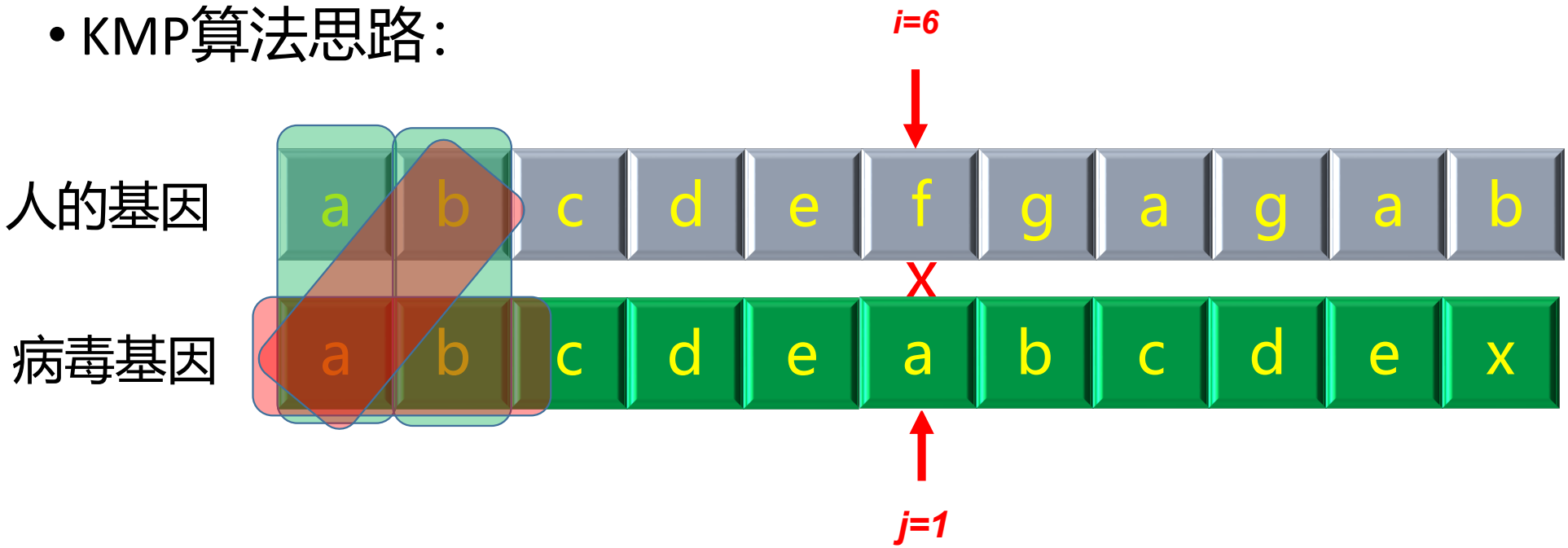
$j = 1$

J指针回溯：从头开始

i指针回溯到这轮开始位置（当前位置—走的步数）的下一个位置

串的模式匹配——KMP算法

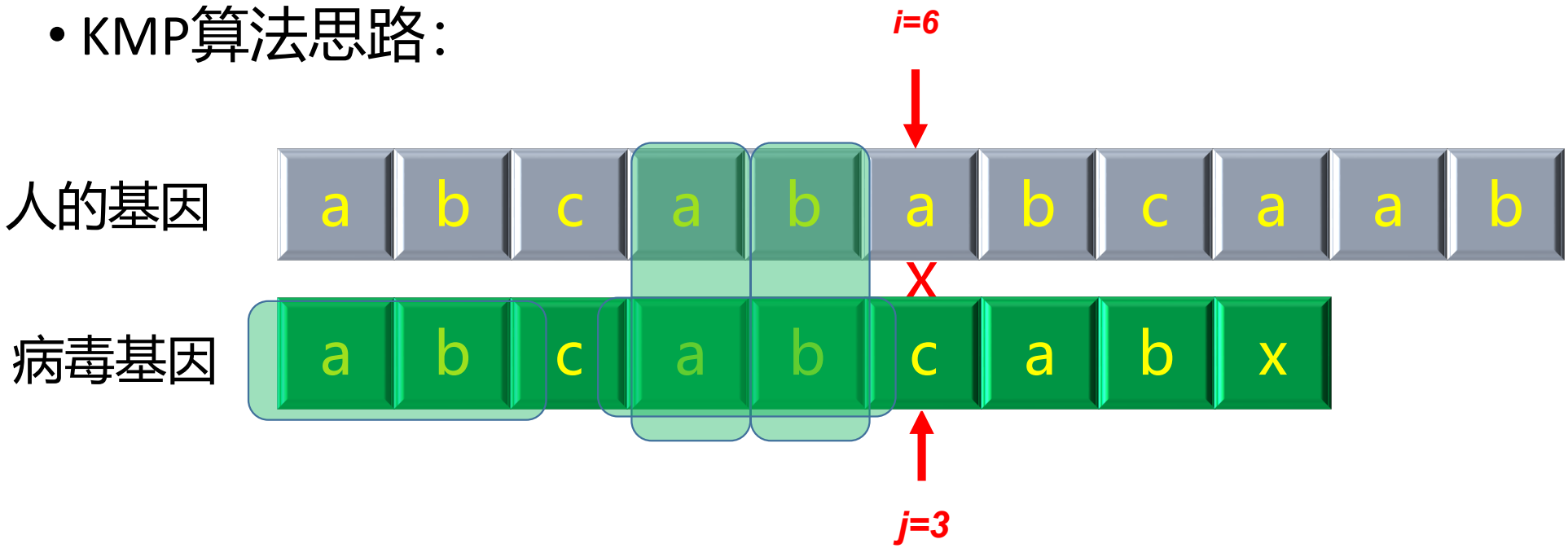
- KMP算法思路：



算法的思路是利用部分匹配的结果加速模式串的滑动速度，主串的i指针不需要回溯，子串的j指针也不一定回溯到头。

串的模式匹配——KMP算法

- KMP算法思路：



算法的思路是利用部分匹配的结果加速模式串的滑动速度，主串的指针不需要回溯，子串的指针也不需要回到头。

KMP算法

杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

```
int Index_KMP(SString S, SString T, int pos){
        i= pos, j = 1;
        while (i<=S.length && j<=T.length) {
                if (j==0 || S.ch[i]==T.ch[j]) {  i++; j++;  }
                else
                   j=next[j];                /*i不变, j后退*/
        }
        if (j>T.length)  return i-T.length;          /*匹配成功*/
        else  return 0;                /*返回不匹配标志*/
}
```

串的模式匹配——KMP算法

- 子串的j指针的回溯，通过next[j]计算。

从头开始的k-1个元素　　j前面的k-1个元素

$$next[j]=\begin{cases}0 & \text{当 } j=1 \text{ 时}\\ \text{Max}\{k \mid 1<k<j \text{ 且 } 'p_1\cdots p_{k-1}'='p_{j-k+1}\cdots p_{j-1}'\} & \text{当此集合不空时}\\ 1 & \text{其他情况}\end{cases}$$

- next数组的前两位直接填0和1
- next[j]表示子串各个位置的 j 值的变化，next[j]只与子串（模式串）相关，与主串无关。

单选题 1分

设置

HANGZHOU DIANZI UNIVERSITY

已知串S=“aaab”，其next数组值为（）

A 0123

B 0112

C 0231

D 1211

《数据结构》

第4-5章 串、数组和广义表

杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

4.1 串的定义

4.2 案例引入

4.3 串的类型定义、存储结构及其运算

5.1 数组

5.2 广义表

5.1 数组

数组：按一定格式排列起来的具有相同类型的数据元素的集合。

一维数组：若线性表中的数据元素为非结构的简单元素，则称为一维数组。

一维数组的逻辑结构：线性结构。定长的线性表。

声明格式：　数据类型　　变量名称[长度];

例: int num[5] = {0，1，2，3，4};

5.1 数组

$$A=(a_0,a_1,\cdots,a_p) \qquad (p=m-1\text{或}n-1)$$

$$a_i=(a_{i0},a_{i1},\cdots,a_{i,n-1}) \qquad 0\leqslant i\leqslant m-1$$

$$a_j=(a_{0j},a_{1j},\cdots,a_{m-1,j}) \qquad 0\leqslant j\leqslant n-1$$

二维数组: 若一维数组中的数据元素又是一维数组结构，则称为二维数组。

二维数组的逻辑结构

- 非线性结构：每一个数据元素既在一个行表中，又在一个列表中。
- 线性结构：定长的线性表
  - 该线性表的每个数据元素也是一个定长的线性表。

5.1 数组

声明格式: 数据类型 变量名称[行数][列数];

例: int num[5] [8];

在C语言中，一个二维数组类型也可以定义为一维数组类型(其分量类型为一维数组类型)，即:

```
typedef elemtype array2[m][n];
```

等价于:

```
typedef elemtype array1[n];
typedef array1 array2[m];
```

5.1 数组

三维数组：若二维数组中的元素又是一个一维数组，则称作三维数组。

.
.
.

n维数组：若n-1维数组中的元素又是一个一维数组结构，则称作n维数组。

结论

线性表结构是数组结构的一个特例，

而数组结构又是线性表结构的扩展。

数组特点：结构固定——定义后，维数和维界不再改变。

数组基本操作：除了结构的初始化和销毁之外，只有取元素和修改元素值的操作。

5.1.1 数组的抽象数据类型定义

n维数组的抽象数据类型

n为数组的维数
b_i为数组第i维的长度

j_i为数组元素第i维的下标

ADT Array{

数据对象：$j_i = 0, \cdots, b_i - 1, i = 1,2, \cdots, n$

D $= \{a_{j_1 j_2 \cdots j_n} | n(>0)$称为数组的维数，$b_i$是数组第$i$维的长度，$j_i$是数组元素的第$i$维下标, $a_{j_1 j_2 \cdots j_n} \in ElemSet\}$

数据关系：$R = \{R1, R2, \cdots, Rn\}$ $Ri = \{< a_{j_1 \cdots j_i \cdots j_n}, a_{j_1 \cdots j_i+1 \cdots j_n} > |$
$0 \le j_k \le b_k - 1, 1 \le k \le n$ 且 $k \ne i$,
$0 \le j_i \le b_i - 2$,
$a_{j_1 \cdots j_i \cdots j_n}, a_{j_1 \cdots j_i+1 \cdots j_n} \in D, i = 2, \cdots, n\}$

5.1.1 数组的抽象数据类型定义

杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

例: 二维数组的抽象数据类型的数据对象和数据关系的定义

n=2(维数为2，二维数组)

b_1: 第1维长度(行数)　　b_2: 第2维长度（列数)

a_{j1j2}: 第1维下标为j_1，第2维下标为j_2

$$\begin{bmatrix} a_{00} & a_{01} & \cdots & a_{0(b_2-1)} \\ a_{10} & a_{11} & \cdots & a_{1(b_2-1)} \\ \vdots & \vdots & a_{j_1j_2} & \vdots \\ a_{(b_1-1)0} & a_{(b_1-1)1} & \cdots & a_{(b_1-1)(b_2-1)} \end{bmatrix}$$

数据对象:

$D = \{a_{ij} \mid 0 \le j_1 \le b_1 - 1,\ 0 \le j_2 \le b_2 - 1\}$

数据关系:

$R = \{ ROW, COL \}$

$COL = \{< a_{j1,j2}, a_{j1+1,j2} > \mid 0 \le j_1 \le b_1 - 2,\ 0 \le j_2 \le b_2 - 1\}$

$ROW = \{< a_{j1,j2}, a_{j1,j2+1} > \mid 0 \le j_1 \le b_1 - 1,\ 0 \le j_2 \le b_2 - 2\}$

5.1.1 数组的抽象数据类型定义

杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

基本操作:

(1) InitArray (&A,n,bound1, ...boundn) //构造数组A

(2) DestroyArray(&A) //销毁数组A

(3) Value(A,&e,index1,...indexn) //取数组元素值

(4) Assign (&A,e,index1,...,indexn) //给数组元素赋值

}ADT Array

5.1.1 数组的顺序存储

因为
- 数组特点: 结构固定——维数和维界不变。
- 数组基本操作: 初始化、销毁、取元素、修改元素值。

一般不做插入和删除操作。

所以: 一般都是采用顺序存储结构来表示数组。

注意: 数组可以是多维的，但存储数据元素的内存单元地址是一维的，因此，在存储数组结构之前，需要解决将多维关系映射到一维关系的问题。

一维数组

例，有数组定义： int a[5];

每个元素占用4字节，假设a[0]存储在2000单元，a[3]地址是多少?

LOC(0)=a=2000　　L=4

LOC(3)=?

LOC(i)=?　　a+i*L

$$LOC(i)=\begin{cases} LOC(0)=a, & i=0 \\ LOC(i-1)+L = a+i*L, & i>0 \end{cases}$$

二维数组

$A=(\alpha_1, \alpha_2, \cdots, \alpha_p)$ (p=m或n)

$\alpha_i=(a_{i1}, a_{i2}, \cdots, a_{in})\ 1 \le i \le m$

$$A_{m\times n}=\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}$$

$$A_{m\times n}=\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}$$

$\alpha_j=(a_{1j}, a_{2j}, \cdots, a_{mj})\ 1 \le j \le n$

两种顺序存储方式
- 以行序为主序(低下标优先)
- 以列序为主序(高下标优先)

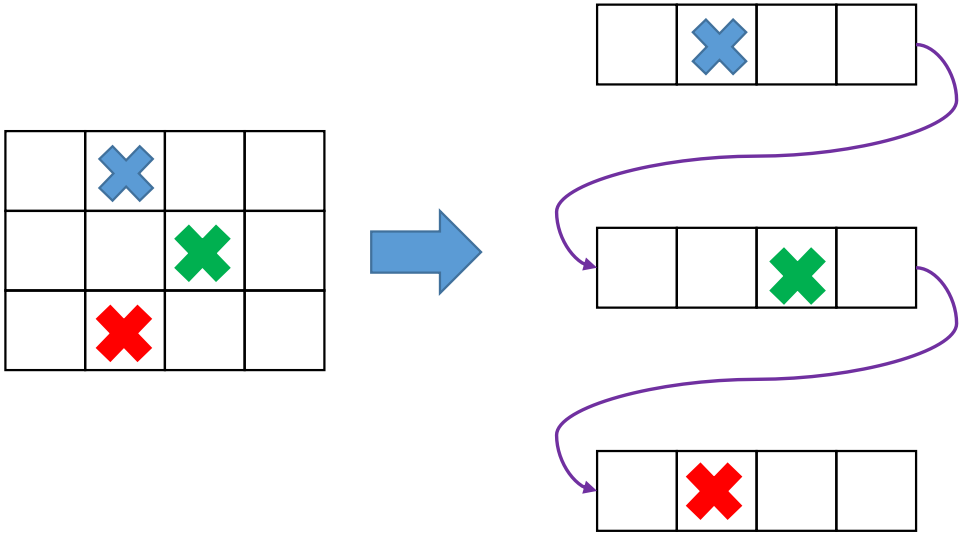
5.1.2 数组的顺序存储

存储单元是一维结构，而数组是个多维结构，则用一组连续存储单元存放数组的数据元素就有个次序约定问题。

- **以行序为主序**

  C，PASCAL，

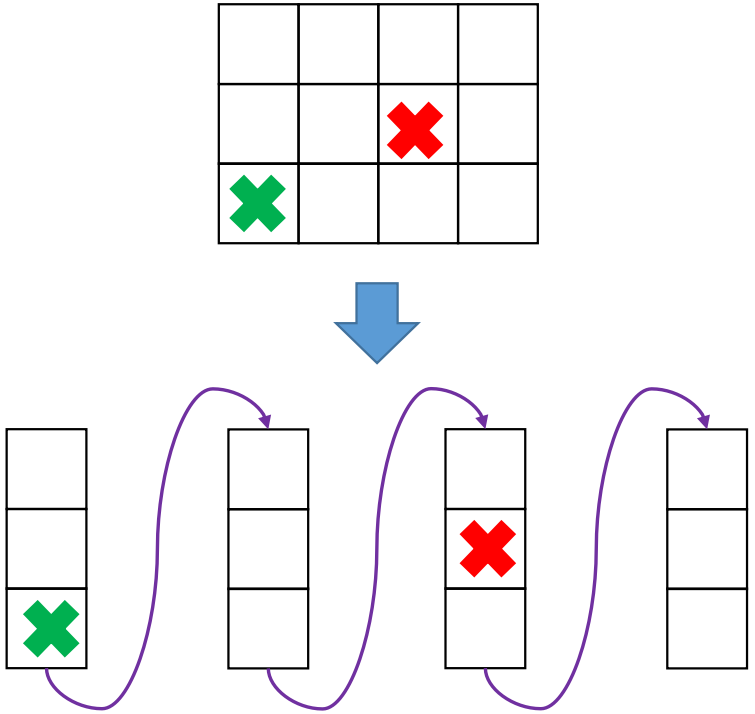
  JAVA，Basic



5.1.2 数组的顺序存储

- **以列序为主序**

  FORTRAN



5.1.2 数组的顺序存储

二维数组可有两种存储方式：

1）以行序为主序；

2）以列序为主序。

$$\begin{bmatrix} a_{00} & a_{01} & \cdots & a_{0,n-1} \\ a_{10} & a_{11} & \cdots & a_{1,n-1} \\ \cdots & \cdots & \cdots & \cdots \\ a_{m-1,0} & a_{m-1,1} & \cdots & a_{m-1,n-1} \end{bmatrix}$$

a_{00}	0
a_{01}	
……	
$a_{0,n-1}$	
a_{10}	
a_{11}	
……	
$a_{1,n-1}$	
……	
$a_{m-1,0}$	
$a_{m-1,1}$	
……	
$a_{m-1,n-1}$	m*n-1

5.1.2 数组的顺序存储

杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

二维数组可有两种存储方式：

1）以行序为主序；

2）以列序为主序。

0	a_{00}
	a_{10}
	……
	$a_{m-1,0}$
	a_{01}
	a_{11}
	……
	$a_{m-1,1}$
	……
	$a_{0,n-1}$
	$a_{1,n-1}$
	……
m*n-1	$a_{m-1,n-1}$

$$\begin{bmatrix} a_{00} & a_{01} & \cdots & a_{0,n-1} \\ a_{10} & a_{11} & \cdots & a_{1,n-1} \\ \cdots & \cdots & \cdots & \cdots \\ a_{m-1,0} & a_{m-1,1} & \cdots & a_{m-1,n-1} \end{bmatrix}$$

a_{00}
a_{01}
……
$a_{0,n-1}$
a_{10}
a_{11}
……
$a_{1,n-1}$
……
$a_{m-1,0}$
$a_{m-1,1}$
……
$a_{m-1,n-1}$

二维数组的行序优先表示

有二维数组A[m][n] (A[0..m-1][0..n-1])

$$A = \begin{pmatrix} a[0][0] & a[0][1] & \cdots & a[0][n-1] \\ a[1][0] & a[1][1] & \cdots & a[1][n-1] \\ a[2][0] & a[2][1] & \cdots & a[2][n-1] \\ \vdots & \vdots & \ddots & \vdots \\ a[m-1][0] & a[m-1][1] & \cdots & a[m-1][n-1] \end{pmatrix}$$

以行序为主序:

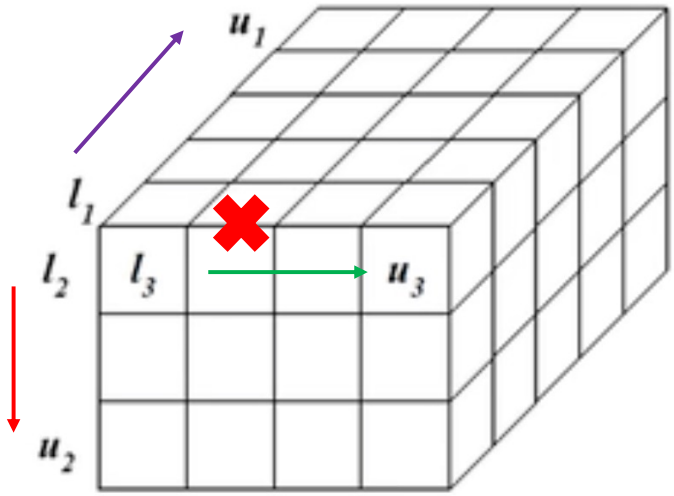
设数组开始存储位置LOC(0, 0) ,存储每个元素需要L个存储单元

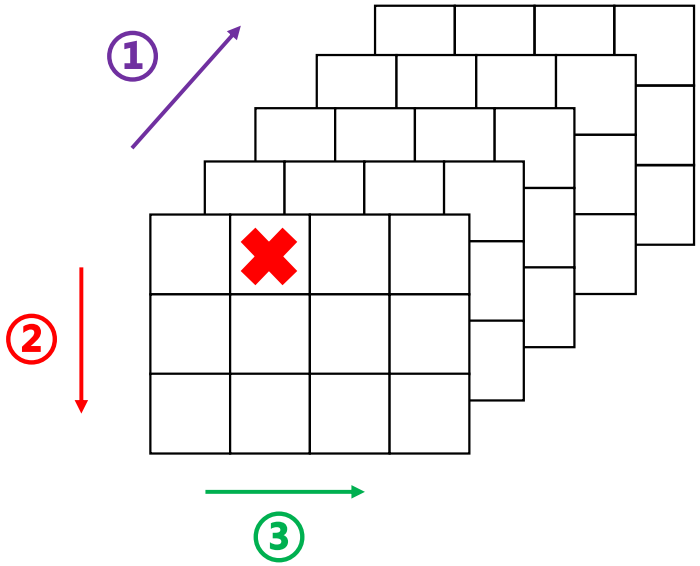
数组元素a[i][j]的存储位置是: LOC(i , j) = LOC(0, 0) + (n*i + j) * L

在a[i][j]前面所有元素个数

三维数组

按页/行/列存放，页优先的顺序存储





三维数组

- a[m_1][m_2][m_3]各维元素个数为m_1, m_2, m_3
- 下标为i_1, i_2, i_3的数组元素的存储位置:

$$LOC(i_1, i_2, i_3) = a + (i_1 * m_2 * m_3 + i_2 * m_3 + i_3) * L$$

- $i_1 * m_2 * m_3$：前 i_1 页总元素个数
- $i_2 * m_3$：第 i_1 页的前 i_2 行总元素个数
- i_3：第 i_2 行前 i_3 列元素个数

n维数组

- 各维元素个数为$m_1, m_2, m_3, .., m_n$
- 下标为$i_1, i_2, i_3, ..., i_n$的数组元素的存储位置:

$$\begin{aligned} LOC(i_1, i_2, \cdots, i_n) &= a + i_1 * m_2 * m_3 * \cdots * m_n + \\ &+ i_2 * m_3 * m_4 * \cdots * m_n + \cdots\cdots + i_{n-1} * m_n + i_n \\ &= a + \left[\left(\sum_{j=1}^{n-1} i_j * \prod_{k=j+1}^{n} m_k \right) + i_n \right] L \end{aligned}$$

n维数组

$$\begin{aligned}\mathrm{LOC}(j_1, j_2, \cdots, j_n) &= \mathrm{LOC}(0, 0, \cdots, 0) + (b_2 \times \cdots \times b_n \times j_1 + b_3 \times \cdots \times b_n \times j_2 \\ &\quad + \cdots + b_n \times j_{n-1} + j_n)L \\ &= \mathrm{LOC}(0, 0, \cdots, 0) + \left(\sum_{i=1}^{n-1} j_i \prod_{k=i+1}^{n} b_k + j_n\right)L\end{aligned}$$

5.1.2 数组的顺序存储

例: 设有一个二维数组A[m][n]按行优先顺序存储，假设A[0][0]存放位置在$644_{(10)}$，A[2][2]存放位置在$676_{(10)}$，每个元素占一个空间，问$A[3][3]_{(10)}$，存放在什么位置? (脚注$_{(10)}$表示用10进制表示。)

设数组元素A[i][j]存放在起始地址为Loc(i, j)的存储单元中

∵ Loc(2,2) = Loc(0,0) + 2 * n + 2 = 644 + 2 * n + 2 = 676.

∴ n = (676 - 2 - 644) / 2 = 15

∴Loc(3,3) = Loc(0,0) + 3 * 15 + 3 = 644 + 45 + 3 = 692

5.1.3 特殊矩阵的压缩存储

杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

矩阵: 一个由 m x n 个元素排成的m行n列的表。

矩阵的常规存储:

将矩阵描述为一个二维数组。

$$A_{m\times n}=\begin{bmatrix} a_{00} & a_{01} & \cdots & a_{0,n-1} \\ a_{10} & a_{11} & \cdots & a_{1,n-1} \\ \vdots & \vdots & & \vdots \\ a_{m-1,0} & a_{m-1,1} & \cdots & a_{m-1,n-1} \end{bmatrix}$$

矩阵的常规存储的特点:

可以对其元素进行随机存取;

矩阵运算非常简单; 存储的密度为1。

不适宜常规存储的矩阵: 值相同的元素很多且呈某种规律分布; 零元素多。

矩阵的压缩存储: 为多个相同的非零元素只分配一个存储空间; 对零元素不分配空间。

5.1.3 特殊矩阵的压缩存储

1.什么是压缩存储?

若多个数据元素的值都相同，则只分配一个元素值的存储空间，且零元素不占存储空间。

2.什么样的矩阵能够压缩?

一些特殊矩阵，如:对称矩阵，对角矩阵，三角矩阵，稀疏矩阵等。

3.什么叫稀疏矩阵?

矩阵中非零元素的个数较少（一般小于5%)

5.1.3 特殊矩阵的压缩存储

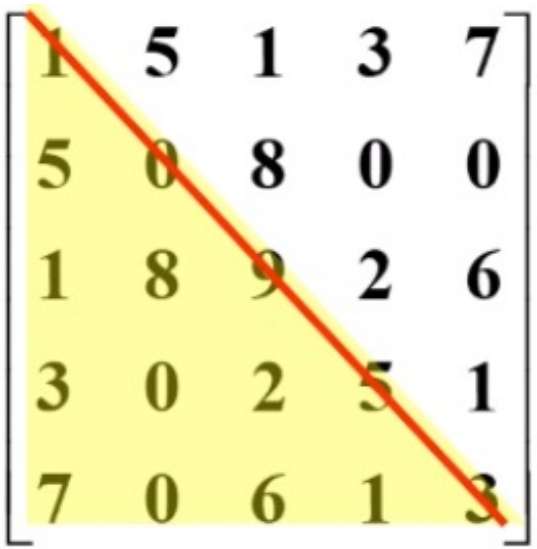
杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

1．对称矩阵

[特点] 在 n x n 的矩阵a中，满足如下性质:

$$a_{ij}=a_{ji}\ (1 \le i, j \le n)$$

[存储方法] 只存储下(或者上)三角(包括主对角线)的数据元素。共占用n(n+1)/2个元素空间。



5.1.3 特殊矩阵的压缩存储

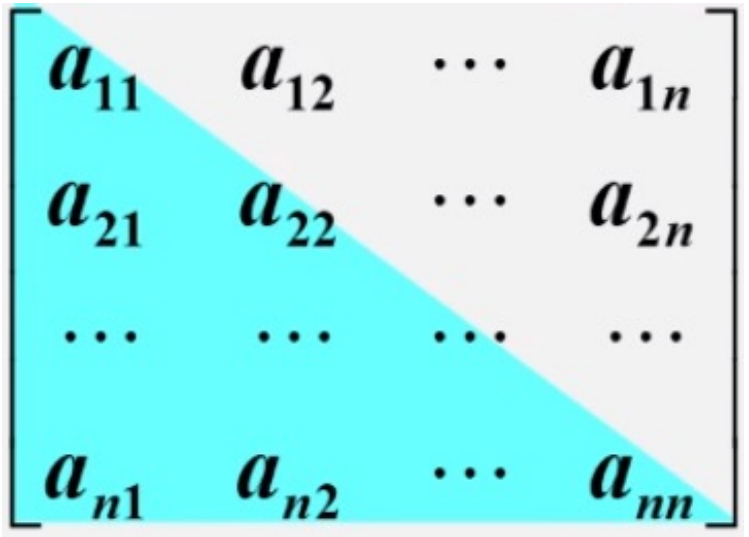
对称矩阵的存储结构:

对称矩阵上下三角中的元素数均为:

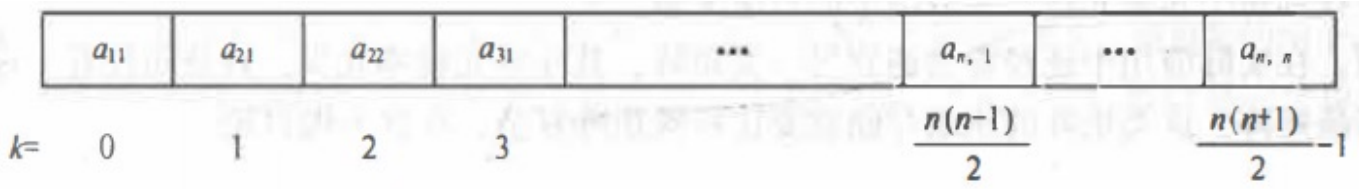
n(n + 1)/2

可以以行序为主序将元素存放在一个一维数组

sa [n(n+1)/2]中。



例如: 以行序为主序存储下三角:

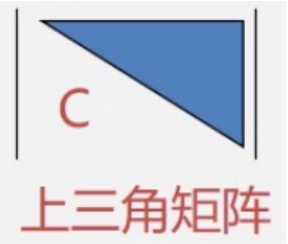


$$\frac{i(i-1)}{2}+j-1$$

5.1.3 特殊矩阵的压缩存储

2. 三角矩阵

[特点] 对角线以下(或者以上)的数据元素(不包括对角线)全部为常数c。



[存储方法] 重复元素c共享一个元素存储空间，共占用n(n+1)/2+1个元素空间: sa[n(n+1)/2+1]

下三角矩阵

$$k=\begin{cases}\dfrac{i(i-1)}{2}+j-1 & 当 i \geq j \\ \dfrac{n(n+1)}{2} & 当\ i<j\end{cases}$$

上三角矩阵

$$k=\begin{cases}\dfrac{(i-1)(2n-i+2)}{2}+(j-i) & 当 i \leq j \\ \dfrac{n(n+1)}{2} & 当\ i>j\end{cases}$$

5.1.3 特殊矩阵的压缩存储

3. 对角矩阵 (带状矩阵)

[特点] 在 n x n的方阵中，所有非零元素都集中在以主对角线为中心的带状区域中，区域外的值全为0，则称为对角矩阵。常见的有三对角矩阵、五对角矩阵、七对角矩阵等。

一个7x7的三对角矩阵

$$\begin{bmatrix} a_{00} & a_{01} & 0 & 0 & 0 & 0 & 0 \\ a_{10} & a_{11} & a_{12} & 0 & 0 & 0 & 0 \\ 0 & a_{21} & a_{22} & a_{23} & 0 & 0 & 0 \\ 0 & 0 & a_{32} & a_{33} & a_{34} & 0 & 0 \\ 0 & 0 & 0 & a_{43} & a_{44} & a_{45} & 0 \\ 0 & 0 & 0 & 0 & a_{54} & a_{55} & a_{56} \\ 0 & 0 & 0 & 0 & 0 & a_{65} & a_{66} \end{bmatrix}$$

5.1.3 特殊矩阵的压缩存储

[存储方法]

- 以对角线的顺序存储

$$\begin{pmatrix} 8 & 2 & 3 & 0 & 0 & 0 \\ 4 & 2 & 0 & 3 & 0 & 0 \\ 5 & 7 & 7 & 6 & 8 & 0 \\ 0 & 9 & 6 & 9 & 1 & 5 \\ 0 & 0 & 6 & 1 & 4 & 2 \\ 0 & 0 & 0 & 2 & 8 & 3 \end{pmatrix}$$

	1	2	3	4	5	6
-2			3	3	8	5
-1		2	0	6	1	2
0	8	2	7	9	4	3
1	4	7	6	1	8	
2	5	9	6	2		

五对角矩阵

稀疏矩阵存储

稀疏矩阵: 设在 m x n 的矩阵中有t个非零元素。

令　　δ = t/(m*n)

当 δ≤0.05 时称为稀疏矩阵。

$$M=\begin{bmatrix} 0 & 12 & 9 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -3 & 0 & 0 & 0 & 0 & 14 & 0 \\ 0 & 0 & 24 & 0 & 0 & 0 & 0 \\ 0 & 18 & 0 & 0 & 0 & 0 & 0 \\ 15 & 0 & 0 & -7 & 0 & 0 & 0 \end{bmatrix}$$

三元组(i, j, a_{ij})唯一确定矩阵的一个非零元

M由{(1,2,12)，(1,3,9)，(3,1,-3),

(3,6,14), (4,3,24), (5,2,18),

(6,1,15), (6,4,-7)}

和矩阵维数(6, 7)唯一确定。

压缩存储原则: 存各非零元的值、行列位置和矩阵的行列数。

三元组的不同表示方法可决定稀疏矩阵不同的压缩存储方法

稀疏矩阵存储----顺序存储结构

杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

1. 三元组顺序表

	i	j	value
0	6	7	8
1	1	2	12
2	1	3	9
3	3	1	-3
4	3	6	14
5	4	3	24
6	5	2	18
7	6	1	15
8	6	4	-7

$$M=\begin{bmatrix} 0 & 12 & 9 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -3 & 0 & 0 & 0 & 0 & 14 & 0 \\ 0 & 0 & 24 & 0 & 0 & 0 & 0 \\ 0 & 18 & 0 & 0 & 0 & 0 & 0 \\ 15 & 0 & 0 & -7 & 0 & 0 & 0 \end{bmatrix}$$

注意：为更可靠描述，通常再加一个“总体”信息：即总行数、总列数、非零元素总个数。

稀疏矩阵存储

例： 试还原出下列三元组所表示的稀疏矩阵。

i	j	value
6	4	6
1	2	2
2	1	12
3	1	3
4	4	4
5	3	6
6	1	16

$$\begin{pmatrix} 0 & 2 & 0 & 0 \\ 12 & 0 & 0 & 0 \\ 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 4 \\ 0 & 0 & 6 & 0 \\ 16 & 0 & 0 & 0 \end{pmatrix}$$

稀疏矩阵存储

三元组顺序表又称有序的双下标法。

三元组顺序表的优点: 非零元在表中按行序有序存储，因此便于进行依行顺序处理的矩阵运算。

三元组顺序表的缺点: 不能随机存取。若按行号存取某一行中的非零元，则需从头开始进行查找。

稀疏矩阵存储

三元组顺序表类型说明：

```
#define MAXSIZE 12500
typedef struct {
    int i, j;                          //该非零元的行下标和列下标
    ElemType e;
}Triple;
typedef struct {
    Triple data[MAXSIZE+1];  //非零元三元组表，data[0]未用
    int mu, nu, tu;                    //矩阵的行数、列数和非零元个数
}TSMatrix;
```

稀疏矩阵的转置

矩阵转置：指变换元素的位置，把位于（i, j）位置上的元素换到 (j, i)位置上，也就是把元素的行、列互换。

$$A=\begin{bmatrix} 15 & 0 & 0 & 22 & 0 & -15 \\ 0 & 11 & 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 6 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 91 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad B=\begin{bmatrix} 15 & 0 & 0 & 0 & 91 & 0 \\ 0 & 11 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 & 0 \\ 22 & 0 & 6 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ -15 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

稀疏矩阵的转置

采用三元组存储矩阵，求矩阵转置算法思路：

1. A的行数、列数、非零元数目转化成B的列数、行数、非零元数目；
2. 按A的列（B的行）进行循环处理：对A的每一列扫描三元组，找出相应的元素，若找到，则交换其行号与列号，并存储到B的三元组中。

$$A=\begin{bmatrix} 15 & 0 & 0 & 22 & 0 & -15 \\ 0 & 11 & 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 6 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 91 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

i	j	value
1	1	15
1	4	22
1	6	-15
2	2	11
2	3	3
3	4	6
5	1	91

稀疏矩阵存储

采用三元组表存储表示，求稀疏矩阵A的转置矩阵B

```
Status TransposeSMatrix(TSMatrix A, TSMatrix &B ){
    B.mu = A.nu;    B.nu = A.mu;   B.tu = A.tu;
    if (B.tu) {
        q = 1;
        for (col =1; col<=A.nu; ++col)
           for (p = 1; p<=A.tu; ++p)
                 if (A.data[p].j == col) {
                     B.data[q].i = A.data[p].j;  B.data[q].j = A.data[p].i
                     B.data[q].e = A.data[p].e;   ++q;
    }
    return OK;
}
```

时间复杂度：O(nu*tu)

稀疏矩阵存储

普通矩阵转置：

```
for (col =1; col<=nu; ++col)
        for (row = 1; p<=mu; ++row)
            B[col][row] =A[row][col];
```

时间复杂度：
O(mu*nu)

当非零元个数tu和mu*nu同数量级时，算法时间复杂度就为O(mu*nu*nu)

虽然节省了存储空间，但时间复杂度提高了，因此只适用于tu<<mu*nu的情况

稀疏矩阵存储

- 因为A中第一列的第一个非零元素一定存储在B.data[1]，第二列的第一个非零元素在B.data中的位置等于第一列的第一个非零元素在B.data中的位置加上第一列的非零元素的个数。以此类推，因为A中的三元组的存放顺序是先行后列的，对同一行来说，必定先遇到列号小的元素，这样只需扫描一边A.data即可。
- 依次按三元组表A的次序进行转置，转置后直接放到三元组表B的正确位置上。这种转置算法称为快速转置算法。

稀疏矩阵存储

- 为了能将待转置三元组表A中的元素一次定位到三元组表B的正确位置上，需要预先计算以下数据：

① 待转置矩阵每一列中非零元素的个数；

② 待转置矩阵每一列中第一个非零元素在三元组表B中的正确位置。

- 为此，需要附设两个向量：

① num[col]: 表示A中第col列非零元素个数；

② cpot[col]: 表示A中第col列第一个非零元素在三元组表B中的正确位置。

稀疏矩阵存储

- num[col]计算方法:

  将A扫描一遍，对于其中列号为col的元素，给相应的num[col]加1。

- cpot[col]计算方法: $\begin{cases} \text{cpot}[1] = 1; \\ \text{cpot}[\text{col}] = \text{cpot}[\text{col}-1] + \text{num}[\text{col}-1] \quad 2 \le \text{col} \le \text{A.nu} \end{cases}$

i	j	value
1	1	15
1	4	22
1	6	-15
2	2	11
2	3	3
3	4	6
5	1	91

col	1	2	3	4	5	6
num[col]	2	1	1	2	0	1
cpot[col]	1	3	4	5	7	7

稀疏矩阵存储


```
Status FastTransposeSMatrix(TSMatrix A, TSMatrix &B) {
    B.mu = A.nu;   B.nu = A.mu;   B.tu = A.tu;
    if (B.tu) {
        for
        for                                                    零元素个数
        cpo
        for                                                    1];
        for
          col = A.data[p].j;      q = cpot[col];
          B.data[q].i = A.data[p].j;      B.data[q].j = A.data[p].i;
          B.data[q].e = A.data[p].e;      ++cpot[col];
        }
    }
    return OK;
}
```

在A的非零元素个数tu和mu*nu等数量级时，其时间复杂度为O(mu*nu)，和经典算法的时间复杂度相同

时间复杂度：O(nu+tu)

稀疏矩阵的链式存储结构：十字链表

杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

- 十字链表

  优点: 它能够灵活地插入因运算而产生的新的非零元素，删除因运算而产生的新的零元素，实现矩阵的各种运算。

  在十字链表中，矩阵的每一个非零元素用一个结点表示，该结点除了(row, col, value)以外，还要有两个域:

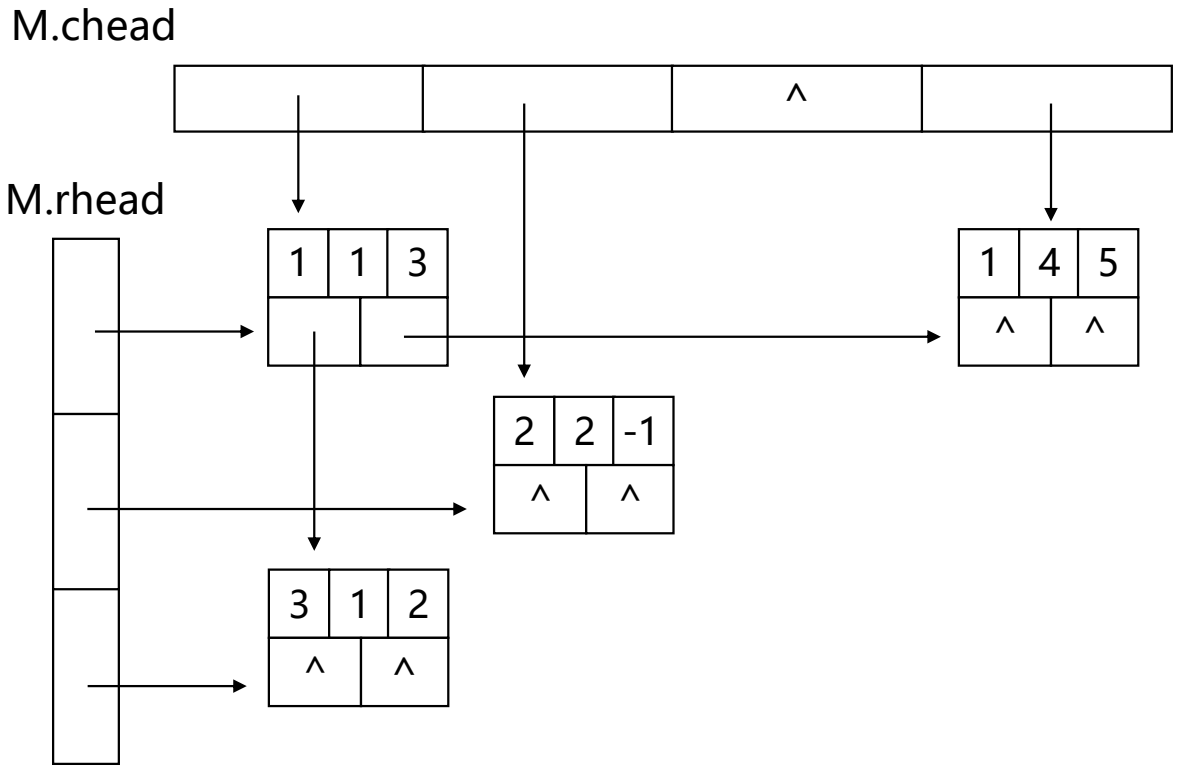
  - right: 用于链接同一行中的下一个非零元素;
  - down: 用以链接同一列中的下一个非零元素。

  十字链表中结点的结构示意图:

row	col	value
down	right	

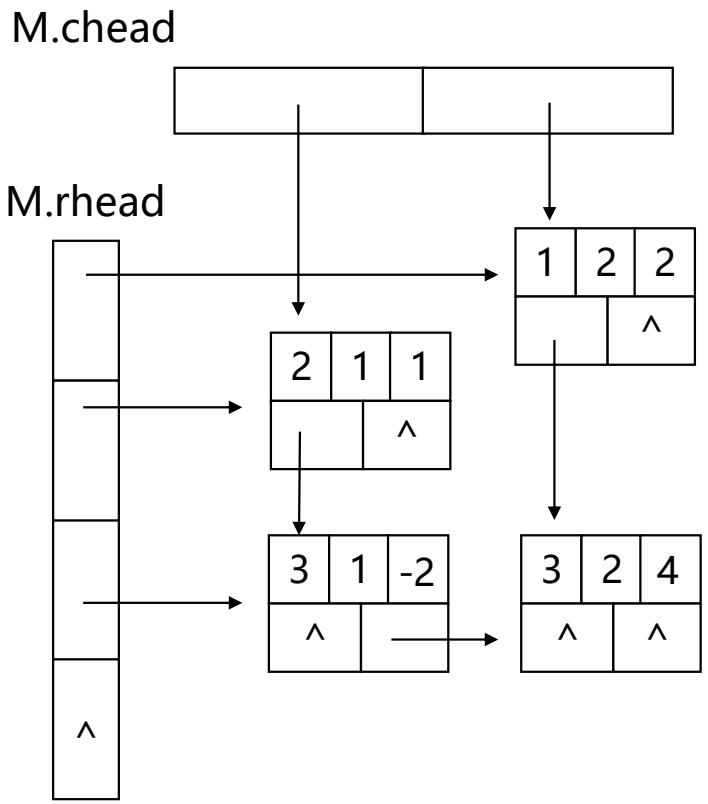
稀疏矩阵的链式存储结构：十字链表

$$M=\begin{bmatrix}3 & 0 & 0 & 5\\ 0 & -1 & 0 & 0\\ 2 & 0 & 0 & 0\end{bmatrix}$$



稀疏矩阵的链式存储结构：十字链表

$$N=\begin{bmatrix} 0 & 2 \\ 1 & 0 \\ -2 & 4 \\ 0 & 0 \end{bmatrix}$$



第4-5章 串、数组和广义表

杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

4.1 串的定义

4.2 案例引入

4.3 串的类型定义、存储结构及其运算

5.1 数组

5.2 广义表

5.2 广义表

广义表(又称列表Lists)是 n≥0 个元素a_0, a_1，…. a_{n-1}的有限序列，其中每一个a_i或者是原子，或者是一个广义表。

例: 中国举办的国际足球邀请赛，参赛队名单可表示如下:

(阿根廷，巴西，德国，法国，( )，西班牙，

意大利，英国，(国家队，山东鲁能，广州恒大))

在这个表中，叙利亚队应排在法国队后面，但未能参加，成为空表。国家队，山东鲁能，广州恒大均作为东道主的参赛队参加，构成一个小的线性表，成为原线性表的一个数据元素。这种拓宽了的线性表就是广义表。

5.2 广义表

- 广义表通常记作: $LS=(a_1, a_2, \ldots, a_n)$

  其中: LS为表名，n为表的长度，每一个a_i为表的元素。

- 习惯上，一般用大写字母表示广义表，小写字母表示原子。

- 表头: 若LS非空($n \geq 1$)，则其第一个元素a_1就是表头。

  记作 $head(LS) = a_1$。 注: 表头可以是原子，也可以是子表。

- 表尾: 除表头之外的其它元素组成的表。

  记作$tail(LS) = (a_2, \ldots, a_n)$。

  注: 表尾不是最后一个元素，而是一个子表。

5.2 广义表

例:

(1) A=( )　　空表，长度为0。

(2) B=(( ))　　长度为1，表头、表尾均为( )。

(3) C=(a, (b, c))　　长度为2，由原子a和子表(b, c)构成。
表头为a; 表尾为((b, c))。

(4) D=(x, y, z)　　长度为3，每一项都是原子。表头为x; 表尾为(y, z)。

(5) E=(C, D)　　长度为2，每一项都是子表。表头为C; 表尾为(D)。

(6) F=(a, F)　　长度为2，第一项为原子，第二项为它本身。
表头为a; 表尾为(F)。

F=(a, (a, (a, ...)))

广义表的性质

(1) 广义表中的数据元素有相对次序; 一个直接前驱和一个直接后继

(2) 广义表的长度定义为最外层所包含元素的个数;

如: C=(a, (b, c))是长度为2的广义表。

(3) 广义表的深度定义为该广义表展开后所含括号的层数;

A = (b, c)的深度为1，B=(A, d)的深度为2，C= (f, B, h)的深度为3。

注意:“原子”的深度为0;“空表”的深度为1。

(4) 广义表可以为其他广义表共享; 如: 广义表B就共享表A。

在B中不必列出A的值，而是通过名称来引用，B= (A)。

(5) 广义表可以是一个递归的表。如:F=(a, F)=(a, (a, (a, ...)))

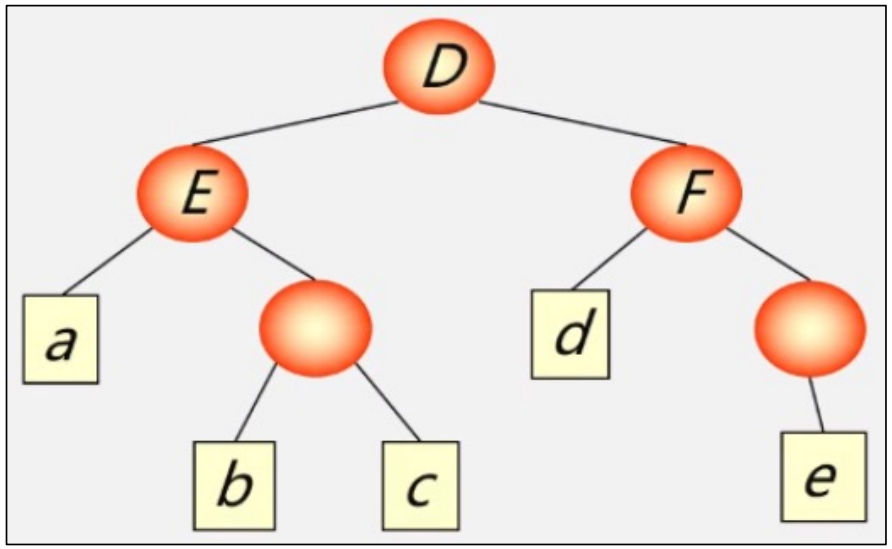
注意: 递归表的深度是无穷值，长度是有限值。

广义表的性质

(6) 广义表是多层次结构，广义表的元素可以是单元素，也可以是子表，而子表的元素还可以是子表，…。

可以用图形象地表示。

例:　　D=(E, F)　　其中:　E=(a, (b, c))　　F=(d, (e))



广义表与线性表的区别?

广义表可以看成是线性表的推广，线性表是广义表的特例。

广义表的结构相当灵活，在某种前提下，它可以兼容线性表、数组、树和有向图等各种常用的数据结构。

当二维数组的每行(或每列)作为子表处理时，二维数组即为一个广义表。

另外，树和有向图也可以用广义表来表示。

由于广义表不仅集中了线性表、数组、树和有向图等常见数据结构的特点，而且可有效地利用存储空间，因此在计算机的许多应用领域都有成功使用广义表的实例。

广义表的基本运算

杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

(1) 求表头GetHead(L): 非空广义表的第一个元素，可以是一个单原子，也可以是一个子表。

(2) 求表尾GetTail(L): 非空广义表除去表头元素以外其它元素所构成的表。表尾一定是一个表。

例: D = (E, F) = ((a, (b, c)), F)

GetHead(D) = E	GetTail(D)= (F)
GetHead(E) = a	GetTail(E) = ((b, c))
GetHead(((b, c))) = (b, c)	GetTail(((b, c))) = ( )
GetHead((b, c)) = b	GetTail((b, c))=(c)
GetHead((c)) = c	GetTail((c))= ()